FIG. 1
PRIOR ART

```
1    c:\collections
2        notes.txt
3        myletter.doc
4        c-myhomepage
5
6          s
7            homepage.html
8            myphoto.jpg
```

FIG. 2

```
1    c:\collections
2        notes.txt
3        myletter.doc
4        c-myhomepage
5          collspec
6          s
7            homepage.html
8            myphoto.jpg
```

100

FIG. 3

```
1    collection       c-myhomepage
2    coll-type        cf-web-page
3    coll-home        cf-colls:mysite.com:c-myhomepage
4    coll-desc        A sample homepage collection
5    end-collection
```

102

FIG. 4

```
c:\collections
    programs
      helloworld
        c-hello-library
        c-hello

    c-myprogram

    parts
      c-include-files
      c-library-one
      c-library-two

    webstuff
      c-myhomepage
```

FIG. 5

```
# Pathnames showing filesystem location of collections

c:\collections\programs\helloworld\c-hello-library
c:\collections\programs\helloworld\c-hello
c:\collections\c-myprogram
c:\collections\parts\c-include-files
c:\collections\parts\c-library-one
c:\collections\parts\c-library-two
c:\collections\webstuff\c-myhomepage
```

FIG. 6
PRIOR ART

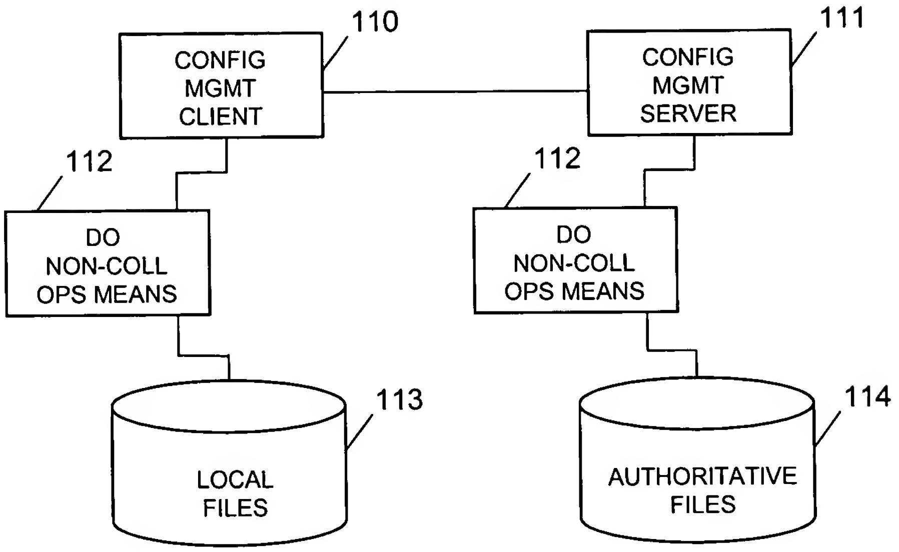

FIG. 7

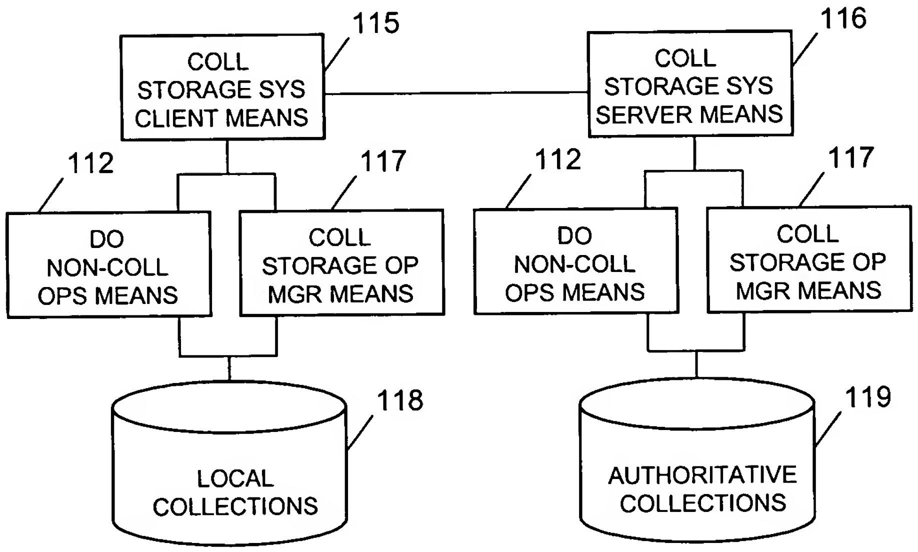
115
COLL STORAGE SYS CLIENT MEANS
116
COLL STORAGE SYS SERVER MEANS
112
DO NON-COLL OPS MEANS
117
COLL STORAGE OP MGR MEANS
112
DO NON-COLL OPS MEANS
117
COLL STORAGE OP MGR MEANS
118
LOCAL COLLECTIONS
119
AUTHORITATIVE COLLECTIONS

FIG. 8

```
# Five components of a full collection reference
#
# <coll-name><scope arguments><content selector names>
# where <coll-name> = category:authority:collection
#
category    the hierarchical category name of the coll
            e.g.  site/prod/release4
authority   the authority name responsible for managing the coll
            e.g.  colls.mysite.com
collection  the collection name
            e.g.  mycoll
scope-args  command line arguments that select scope within coll
            e.g. –recursive –new –changed –locked
selectors   names of categories, directories, or files
            e.g.  mydir, mydir/myfile.c, collspec.txt
```

FIG. 9

```
# Example collection reference for collection of FIG 2

# coll c-myhomepage in cat cf-colls at mysite.com
 cf-colls:mysite.com:c-myhomepage
```

FIG. 10

```
# Shortcut collection references and their meanings
#
# Shortcut        Meaning

cat:auth:coll     a full collection name reference
cat:auth:         all collections in category at authority
cat::coll         this coll in this cat at default authority
cat::             all colls in this cat at default authority
:auth:coll        this coll in all cats at this authority
:auth:            all colls in all cats at this authority

::                current coll if inside a coll; invalid outside a coll
::.               current coll and current dir if inside; invalid outside
::mydir           current coll and mydir if inside; invalid outside
```

FIG. 11

```
# Structure of a collection symbolic job request
#
# "do this (task) to that (collection reference)"
# <task-name> <collection-ref>
# where <collection-ref> = category:authority:collection
#
category    the hierarchical category name of the coll
            e.g.  site/prod/release4
authority   the authority name responsible for managing the coll
            e.g.  colls.mysite.com
collection  the collection name
            e.g.  mycoll
```

FIG. 12

```
# Example collection symbolic job requests
# (using collections from FIGs 2-3)

# rebuild coll c-myhomepage in cat cf-colls at mysite.com
 rebuild cf-colls:mysite.com:c-myhomepage

# rebuild all collections in category cf-colls at mysite.com
rebuild cf-colls:mysite.com:
```

FIG. 13

```
# Expanded collection list for a symbolic job request
#
# <task-name> <collection-ref>
# rebuild cf-colls:mysite.com:
#
cf-colls:mysite.com:c-hello-library
cf-colls:mysite.com:c-hello
cf-colls:mysite.com:c-myprogram
cf-colls:mysite.com:c-include-files
cf-colls:mysite.com:c-library-one
cf-colls:mysite.com:c-library-two
cf-colls:mysite.com:c-myhomepage
```

FIG. 14

```
# Expanded (sorted) visit order list for a symbolic job request
#
# <task-name> <collection-ref>
# rebuild cf-colls:mysite.com:
#
cf-colls:mysite.com:c-include-files     10
cf-colls:mysite.com:c-library-two       49
cf-colls:mysite.com:c-hello-library     50
cf-colls:mysite.com:c-library-one       50
cf-colls:mysite.com:c-hello             100
cf-colls:mysite.com:c-myhomepage        100
cf-colls:mysite.com:c-myprogram         100
```

FIG. 15

```
# Expanded platform list for cf-colls:mysite.com:c-myprogram
# Platform names are user-defined, and are not trademarks
win2000
win98
win95
linux2
```

FIG. 16

```
# Expanded platform list for cf-colls:mysite.com:c-myhomepage
#
win2000
```

FIG. 17

```
# Expanded job triplet list for a single collection
#
cf-colls:mysite.com:c-myprogram    100   win2000
cf-colls:mysite.com:c-myprogram    100   win98
cf-colls:mysite.com:c-myprogram    100   win95
cf-colls:mysite.com:c-myprogram    100   linux2
```

FIG. 18

```
# Expanded job triplet list for a single collection
#
cf-colls:mysite.com:c-myhomepage  100   win2000
```

FIG. 19

```
# Expanded triplet list for a symbolic job request
#
# rebuild cf-colls:mysite.com:

cf-colls:mysite.com:c-include-files     10     win2000
cf-colls:mysite.com:c-include-files     10     win98
cf-colls:mysite.com:c-include-files     10     win95
cf-colls:mysite.com:c-include-files     10     linux2

cf-colls:mysite.com:c-library-two       49     win2000
cf-colls:mysite.com:c-library-two       49     win98
cf-colls:mysite.com:c-library-two       49     win95
cf-colls:mysite.com:c-library-two       49     linux2

cf-colls:mysite.com:c-hello-library     50     win2000
cf-colls:mysite.com:c-hello-library     50     win98
...                                     ...    ...

cf-colls:mysite.com:c-library-one       50     win2000
cf-colls:mysite.com:c-library-one       50     win98
...                                     ...    ...

cf-colls:mysite.com:c-hello             100    win2000
cf-colls:mysite.com:c-hello             100    win98
...                                     ...    ...

cf-colls:mysite.com:c-myhomepage        100    win2000

cf-colls:mysite.com:c-myprogram         100    win2000
cf-colls:mysite.com:c-myprogram         100    win98
cf-colls:mysite.com:c-myprogram         100    win95
cf-colls:mysite.com:c-myprogram         100    linux2
```

FIG. 20

```
/* data structure for holding expanded job info */

csje-info {
  + original symbolic task name
  + original collection-reference

  + expanded-coll-list

      + coll-structure-1
          + coll-name
          + visit-order
          + platform-list
            + platform-1
            + platform-2
            + platform-...
          + other-coll-info

      + coll-structure-2
          + coll-name
          + visit-order
          + platform-list ...
          ...

      + coll-structure-3
          ...

  + other expansion info
}
```

FIG. 21

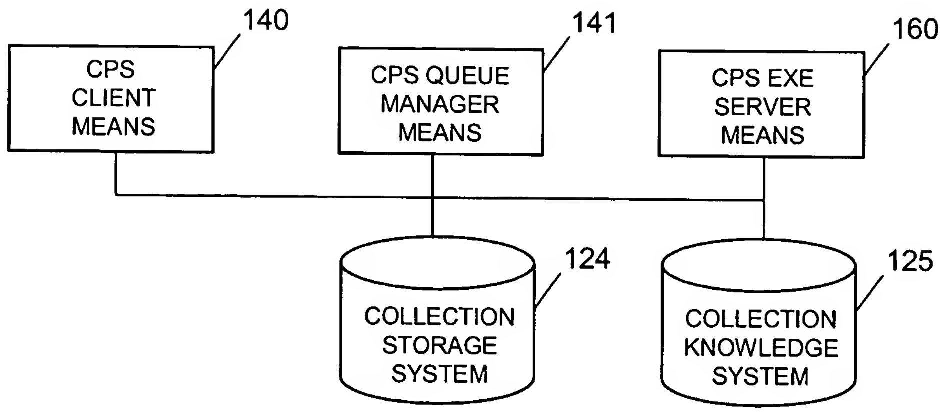

FIG. 22

```
1   # Simplified pseudocode algorithm for a CPS system
2   #
3   CPS Client receives symbolic job request, passes it to Queue Manager
4   CPS Queue Manager expands symbolic request into detailed job requests
5   CPS Queue Manager dispatches detailed requests to exe servers
6   CPS Execution Servers perform requested computations
7   CPS Queue Manager reports job results to request originator
8   All modules may make use of CSS 124 and CKS 125 systems
```

FIG. 23

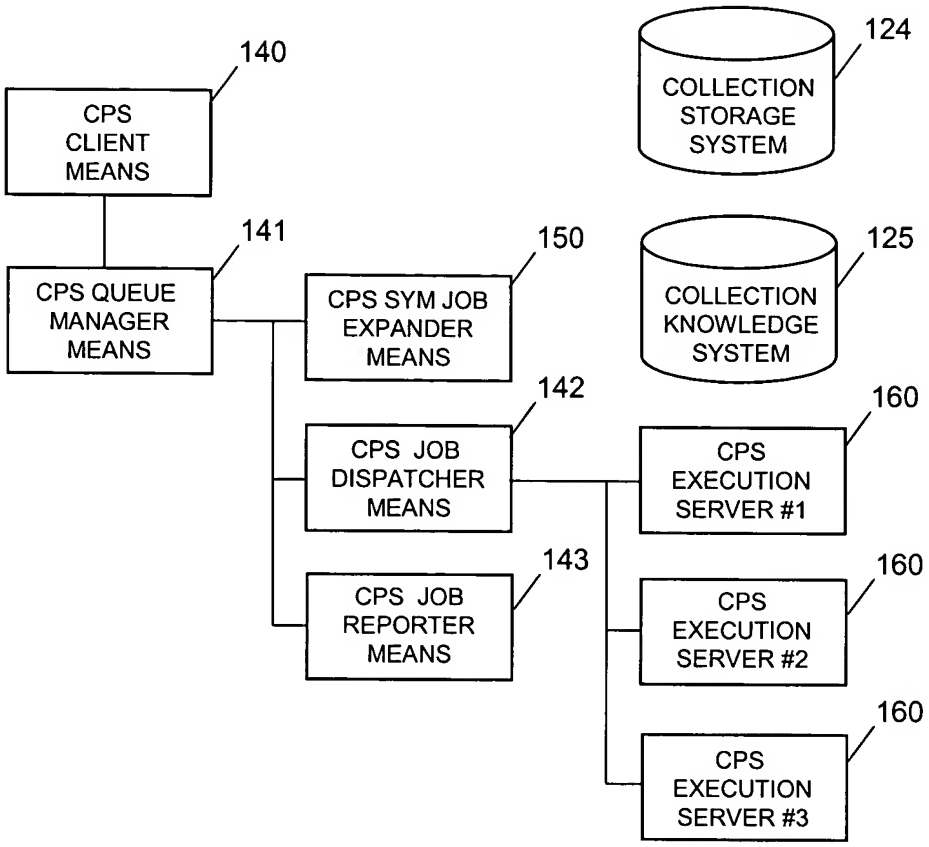

FIG. 24

```
1   # Simplified pseudocode algorithm for a collection processing system
2   CPS client receives job request, passes it to Queue Manager
3   Queue mgr calls internal job expander to expand job request
4   Queue mgr calls job dispatcher to expand and dispatch job triplets
5   Job dispatcher distributes second-level cmds among N exe servers
6   Job reporter means aggregates and reports job results
7   All modules may use CSS 124 and CKS 125, as required
```

FIG. 25

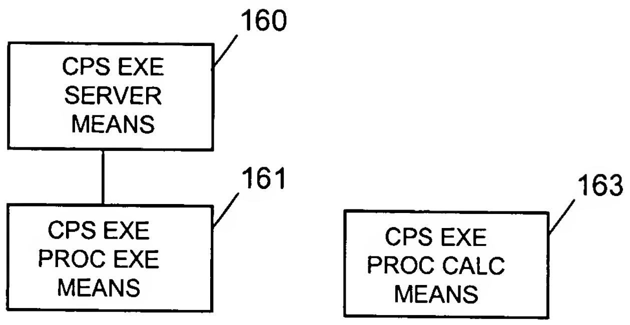

FIG. 26

```
# Simplified pseudocode algorithm for a CPS execution server
#
Receive 2nd level prologue/main/epilogue cmd from job dispatcher
Expand 2nd level command into 3rd level command
Call CPS Exe Process Execution Means to execute 3rd level cmd
Report job results back to CPS Job Dispatcher
```

FIG. 27

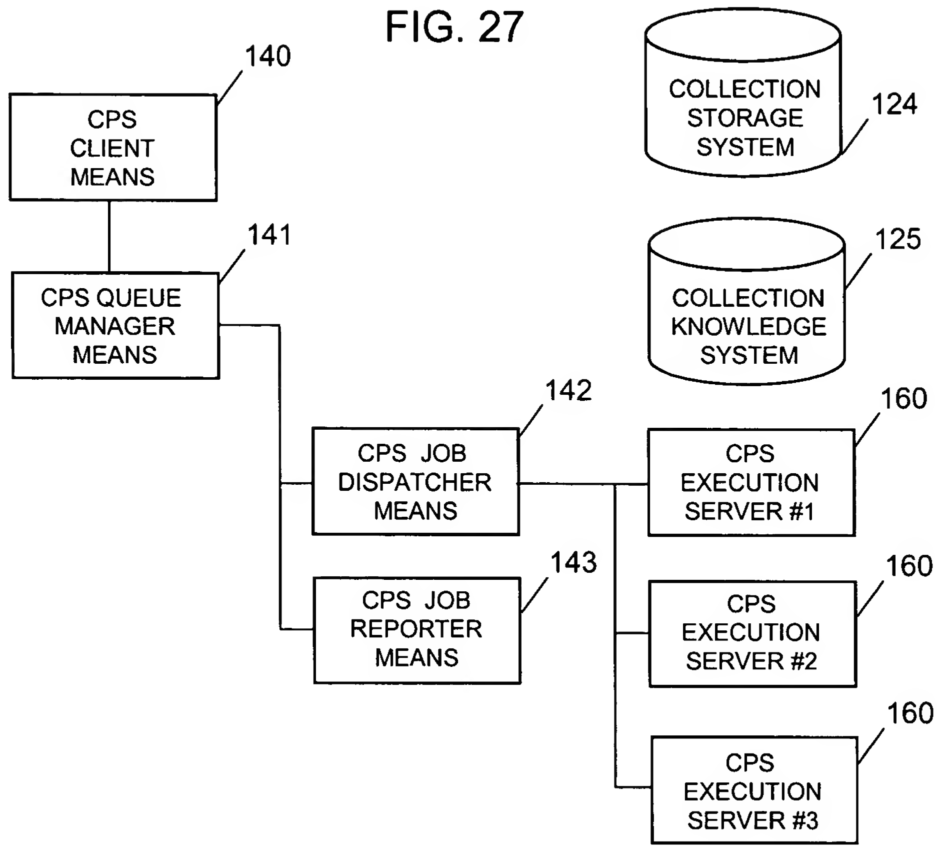

FIG. 28

```
1   # Simplified pseudocode algorithm for a collection processing system
2   CPS client receives job request, passes it to Queue Manager
3   Queue mgr calls external job expander to expand job request
4   - Q mgr calls job dispatcher, job dispatcher calls exe server
5   - Exe server calls external job expander program
6   - External job expander returns list of expanded job triplets
7   Queue mgr calls job dispatcher to expand and dispatch job triplets
8   Job dispatcher distributes second-level cmds among N exe servers
9   Job reporter means aggregates and reports job results
10  All modules may use CSS 124 and CKS 125, as required
```

FIG. 29

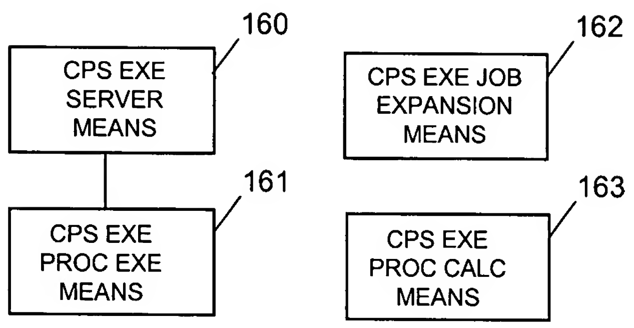

FIG. 30

```
# Simplified pseudocode algorithm for a CPS execution server
# that uses external program for job expansion
#
Receive 2nd level cmd from job dispatcher
If cmd is a job expansion cmd, call CPS exe job expansion means
- return expanded job triplets back to CPS Queue Manager
If cmd is a 2nd level prologue/main/epilogue cmd
- Expand 2nd level command into 3rd level command
- Call CPS Exe Process Execution Means to execute 3rd level cmd
Report job results back to CPS Job Dispatcher
```

FIG. 31

```
# syntax of a task definition file

taskpart <taskpart-name> [options] [attributes]

where
taskpart            - a label to make parsing easier
taskpart-name       - symbolic taskpart name

options             - task command options
    sync            - wait until all platforms finish
    one             - do command on only one platform
    all             - do command on all platforms

attributes
    fast            - run taskpart on a fast computer
    native          - run taskpart on native platform
                      (for running executables and tests)
    make            - run taskpart using Make interpreter
    perl            - run taskpart using Perl interpreter
    <other>         - other user-defined attrs are possible
```

FIG. 32

```
name-symbolic-task.tbl:
# a table of symbolic task names

# name              definition file
rebuild             task-rebuild.def
export              task-export.def
regtest             task-reg-test.def
```

FIG. 33

```
task-rebuild.def:
# a task definition file for rebuilding a c program collection

# checkout coll from collection storage system using a fast cpu
# to avoid checkout collisions, only one platform does checkout
taskpart checkout fast one

# all platforms must wait until the checkout is done
taskpart sync sync

# generate a custom makefile containing third-level cmd seqs
taskpart gen-makefile

# compile and link the program locally
taskpart local make

# export built platform-independent files to a team shared tree
# to avoid collisions, one platform exports pi files first
taskpart export make one

# all platforms must wait until the export is done
taskpart sync sync

# now all other platforms can export
# no collisions now, since timestamps on pi files are up to date
taskpart export make

# epilogue commands
# whatever else users define
```

FIG. 34

```
cps-exe-vars.tbl:
# a table of variables defined by cps execution servers

REF             reference directory holding exe workspace
CAT             category part of collection reference
AUTH            authority part of collection reference
COLL            collection name part of collection reference
PLT             current platform of exe server
CMD-DIR         default directory to execute commands in
MAKE            name of make program on PLT
...
```

FIG. 35

```
# example substitution string values for win2000 platform

REF                 c:\cps\work
CAT                 cf-colls
AUTH                mysite.com
COLL                c-program
PLT                 win2000
MAKE                nmake
```

FIG. 36

```
# example substitution string values for linux2 platform

REF                 /usr/local/cps/work
...
PLT                 linux2
MAKE                make
```

FIG. 37

```
name-task-part.tbl:
# a table of symbolic task PART names

# name              definition file
checkout            taskpart-checkout.def
gen-makefile        taskpart-gen-makefile.def
local               taskpart-local.def
export              taskpart-export.def
```

FIG. 38

```
taskpart-checkout.def:
# a task part definition file for checking out a collection
# syntax:
#   var <var-name> <var-value>
#   cmd <command-to-execute>

# all commands are run in directory $(CMD-DIR)
# variable substitutions are performed before execution

# make a directory in which to checkout the collection
cmd mkdir $(REF-CAT-DIR)

# set CMD-DIR to run all further commands in the new dir
var CMD-DIR $(REF-CAT-DIR)

# checkout a collection from a collection storage system (css)
cmd css checkout $(CAT):$(AUTH):$(COLL)
```

FIG. 39

```
taskpart-gen-makefile.def:
# a task part definition file for generating a makefile
#   cmd <command-to-execute>
#   var <var-name> <var-value>
# all commands are run in directory $(CMD-DIR)

# make a platform dir, execute commands in there
cmd mkdir $(PLT)
var CMD-DIR $(PLT)

# generate a makefile using the 'smnow' generator program
cmd smnow go
```

FIG. 40

```
taskpart-local.def:
# a task part definition file for building coll locally
#   cmd</platform>    <command-to-execute>
#   var</platform>    <var-name> <var-value>
# all commands are run in directory $(CMD-DIR)

# make a platform dir, execute commands in there
cmd mkdir $(PLT)
var CMD-DIR $(PLT)

# generate a makefile using the 'smnow' generator program
cmd clean make
cmd local make
```